Gestion contextualisée de la sécurité : implémentation MDS@Runtime avec FraSCAti

Wendpanga Francis Ouedraogo, Frédérique Biennier, Philippe Merle

Gestion contextualisée de la sécurité : implémentation MDS@Runtime avec FraSCAti

Wendpanga Francis Ouedraogo
(wendpanga-francis.ouedraogo@liris.cnrs.fr)*
Frédérique Biennier (frederique.biennier@liris.cnrs.fr)*
Philippe Merle (philippe.merle@inria.fr)†

Résumé : The development of security policies for information systems is usually based on a systematic risks analysis, reducing them by adopting appropriate countermeasures. These risks analysis approaches are complex and designed for well-known and static environments. To overcome this limit, we propose to extend the Model Driven Security (MDS) approach to a MDS@Runtime vision to set a Security as a Service component. Plugged on the FraSCAti middleware, our security component selects, composes and orchestrates the security services depending on the execution context to avoid both under and over protection.

Mots Clés : MDS@Runtime, SecaaS, processus métiers, Cloud Computing, SCA, FraSCAti

1 Introduction

Pour répondre aux besoins d'agilité et de flexibilité induits par les fortes évolutions du monde économique, les entreprises développent de plus en plus des stratégies de collaboration induisant une ouverture de leur système d'information (SI) et la construction de processus métiers collaboratifs interconnectant ces SI. Pour répondre aux contraintes d'agilité et d'interopérabilité, on peut recourir à des outils permettant de composer ces processus collaboratifs en réutilisant des services métiers puis déployer les processus obtenus sur des infrastructures en nuage. Si une telle stratégie augmente l'agilité du SI, elle n'est pas sans poser d'évidents problèmes de sécurité. Le développement de stratégies de sécurité est usuellement basé sur une analyse systématique des risques afin de les réduire en adoptant des contre-mesures adaptées. Ces démarches méthodologiques sont lourdes, complexes à mettre en œuvre et sont souvent rendues caduques car les risques sont évalués dans un monde « fermé », ce qui n'est pas le cas d'une approche par composition de services métiers réutilisables où le contexte d'utilisation des différents services au niveau métier et plateforme est inconnu a priori. C'est donc sur la base de l'analyse du contexte que l'on peut définir les contraintes de sécurité propres à chaque service métier et déployer les moyens de sécurisation adaptés pour respecter ces politiques de protection. C'est pour répondre à cet enjeu que nous proposons dans ce papier d'étendre l'ingénierie de la sécurité dirigée par les modèles (MDS) vers une logique de MDS@Runtime, c'est-à-dire que

*. Université de Lyon, CNRS INSA-Lyon, LIRIS UMR 5205, 20 avenue Albert Einstein, 69621 Villeurbanne Cedex, France.
†. Inria Lille - Nord Europe, Parc Scientifique de la Haute Borne, 40 avenue Halley, 59650 Villeneuve d'Ascq, France.

les modèles de sécurité sont encore présents à l'exécution, et de gérer la sécurité comme un service pluggable sur FraSCAti, un intergiciel permettant de supporter des processus collaboratifs.

2 État de l'art

2.1 Intergiciel comme PaaS

L'implémentation d'une architecture orientée service (SOA) nécessite un intergiciel (middleware en anglais) pour à la fois l'intégration et la communication distribuée. L'intergiciel joue un rôle de médiateur entre le client et le fournisseur de services. Il s'agit d'un composant logiciel qui se place entre le système d'exploitation et les applications métiers et offre un haut niveau d'abstraction pour la construction d'applications distribuées. Il permet l'intégration et la gestion des services d'entreprise et fournit un accès aux différents services externes [SHLP05]. C'est une solution d'intégration implémentant une architecture totalement distribuée (déploiement sur plusieurs nœuds), fournissant des services comme la transformation des données ou le routage basé sur le contenu (CBR), ainsi qu'un niveau d'interopérabilité accru par l'utilisation systématique des standards comme XML, les Web Services et les normes WS-* [Lou08]. FraSCAti [1] [SMF+09, SMR+12] est un intergiciel se basant sur le standard OASIS Service Component Architecture (SCA) pour construire des applications métiers orientées services et adaptables. Les applications en FraSCAti peuvent être déployées dans différents nuages (Amazon EC2, Amazon Elastic BeanTalk, Google App Engine, CloudBees, etc.) [MRS11, PHM+12]. L'adaptabilité à la conception repose sur le fait que la plateforme FraSCAti a été conçue comme une architecture à base de plugins pour pouvoir s'adapter à différents environnements d'exécution et pouvoir sélectionner à la demande les fonctionnalités dont on a besoin pour composer sa plateforme. L'adaptation à l'exécution repose sur les fonctionnalités réflexives de FraSCAti, c'est-à-dire des capacités d'introspection et de reconfiguration des applications durant l'exécution. Un tel support d'exécution permet de rendre un SI plus agile et reconfigurable. Toutefois ceci pose d'évidents problèmes de sécurité.

2.2 MDS pour la sécurisation de processus métiers

La sécurité dirigée par les modèles (MDS) [LS09] est une approche qui décrit le processus de modélisation des exigences de sécurité à un haut niveau d'abstraction en utilisant un langage de modélisation spécifique au domaine (DSL). Utilisant différentes sources d'information pour définir les besoins de protection, la stratégie MDS [BDL03, CSBE08] adapte l'approche OMG MDA au domaine de la sécurité en utilisant des modèles de sécurité basés sur UML en général ou des annotations de sécurité plus spécifiques aux processus métiers [SSL+09, WMS+09] pour limiter autant que faire se peut les interventions humaines pour transformer ces besoins en politiques de sécurité efficaces. Afin d'assurer la sécurité des processus métiers, plusieurs travaux basés sur MDS ont vu le jour et ont donnés lieu à des cadriciels (frameworks en anglais) tels que OpenPMF, SECTET et BPSec. Ces cadriciels se basent sur des modèles UML et sur des annotations de sécurité pour définir les besoins de sécurité puis les transformer de manière automatique en politiques

1. `http://frascati.ow2.org`

de sécurité. Cependant, ces cadriciels supposent que la spécification des protections nécessaires a été faite au préalable et ne permettent pas de prendre en compte le contexte d'exécution pour déployer de manière efficace les protections correspondantes aux besoins. Or, pour sécuriser un processus, un analyste métier devrait identifier au préalable le niveau de sécurité à offrir et les contraintes de sécurité à respecter pour chaque activité, service ou donnée du processus avant d'envisager d'en faire les spécifications, opération nécessitant d'analyser les risques auxquels le système est exposé. Les méthodes d'analyse des risques usuellement développées pour l'ingénierie de la sécurité des SI (telles que EBIOS, MEHARI, OCTAVE, etc.) sont complexes et nécessitent de recourir à un expert en sécurité pour les mettre en œuvre. De plus, elles visent à sécuriser un périmètre identifié à l'avance ce qui ne s'adapte pas au contexte ouvert des processus collaboratifs déployés dans des environnements orientés services.

3 MDS@Runtime avec FraSCAti

Pour permettre à des architectes métiers de définir des politiques de sécurité adaptés aux besoins, nous avons proposé dans [OBG12] de coupler la stratégie MDS à l'ingénierie orientée patrons pour réaliser la capture des besoins et de prendre en compte des modèles de plateformes d'exécution pour générer de manière automatique des politiques de sécurité associées aux services métiers (voir figure 1). Pour permettre d'assurer la portabilité et le déploiement contextualisés de ces services sécurisés, nous proposons dans ce papier d'étendre cette stratégie avec une approche Model@Runtime obtenant ainsi ce que nous nommons MDS@Runtime, c'est-à-dire que les modèles de sécurité sont présents à l'exécution et utilisés pour orchestrer les services techniques de sécurité. Notre nouvelle approche a été prototypée au dessus de l'intergiciel FraSCAti.

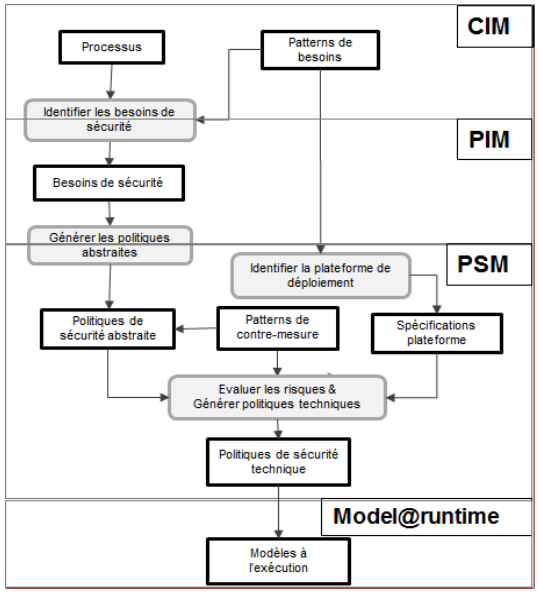

Figure 1: Le processus MDS de génération des politiques de sécurité.

FraSCAti propose un cadre de programmation, de déploiement et d'exécution de composants implantant des services conforme au standard OASIS SCA. Afin d'assurer la sécurité des processus métiers déployés sur des infrastructures en nuage, nous proposons un cadriciel de sécurité à base de composants SCA qui vient se plugger sur la plateforme FraSCAti. Ce cadriciel comprend deux composites, c'est-à-dire des composants contenant des sous-composants.

Le composite **MDS@Runtime** permet d'intercepter les appels de services métiers et d'appeler les services de sécurité qui mettent en application les politiques de sécurité. Ce composite comprend 5 composants principaux (voir figure 2) :

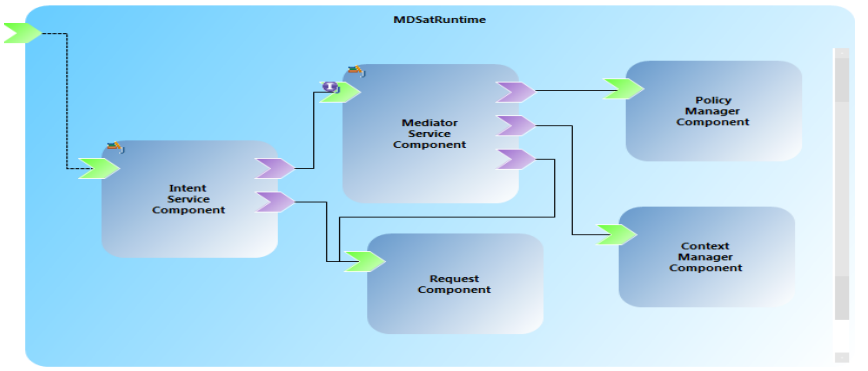

Figure 2: Le composite **MDS@Runtime**.

- Le composant *Intent* est chargé de détecter et d'intercepter les invocations des services métiers initiées par les clients. Ce composant utilise des techniques de la programmation orientée aspect (AOP) mise en œuvre dans FraSCAti afin effectuer des actions avant, pendant et après chaque invocation de services métiers. Ces techniques utilisent les mécanismes d'interception de la pile Apache CXF embarquée dans FraSCAti. Une fois les intentions des clients capturées (par exemple le message Apache CXF généré par un appel du service), il doit être formaté pour être interprété par le composant *Mediator* afin d'assurer la sécurité.
- Le composant *Request* joue le rôle d'intermédiaire entre l'intergiciel FraSCAti et les services de sécurité. Il fournit une interface bidirectionnelle qui permet au composant *Intent* de formaliser les messages d'interaction reçus de la plateforme FraSCAti et aussi d'initier des ordres vers cette dernière afin qu'elle effectue certaines actions techniques. Ce composant assure une indépendance totale entre l'intergiciel FraSCAti et notre système de sécurité, ce qui permet d'une part de pouvoir déployer et exécuter les services de sécurité sur n'importe quelle autre intergiciel et d'autre part de ne déployer sur une plateforme spécifique que les services de sécurité dont on a besoin.
- Le composant *Mediator* est chargé d'analyser les demandes d'appel de services interceptées par le composant *Intent* et encapsulées par le composant *Request* puis d'identifier les règles de politiques de sécurité associées au service métier avec lequel le client a besoin d'interagir. Ainsi, grâce au composant *Request*, le *Mediator* reçoit les informations sur les services impliqués dans l'interaction. Ces informations sont utilisées pour obtenir les politiques associées aux ressources (opérations

qui implémentent les fonctionnalités des services métiers). Ces politiques sont ensuite analysées et orchestrées par le *Mediator* pour appeler les services de sécurité nécessaires.

- Le composant *PolicyManager* assure la gestion des politiques. Il reçoit du *Mediator* la référence de la ressource ou service sollicité ainsi que le lien vers le fichier de politique et retourne au *Mediator* la liste des politiques à appliquer.
- Le composant *ContextManager* analyse les politiques de sécurité associées aux services et identifie les différentes politiques à appliquer selon le contexte de l'utilisateur, l'environnement d'exécution et les politiques de sécurité associées au client et au fournisseur de service. Il fournit également au composant *Mediator* les informations telles que les politiques et règles de politiques correspondantes au contexte d'exécution. Ces règles de politiques sont utilisées par le *Mediator* de sécurité pour appeler les services techniques de sécurité.

Le composite **Security as a Service** regroupe les différents services de sécurité à même d'assurer une protection des ressources et services métiers selon une logique de sécurité "as a service". Ce composite contient les composants suivants (voir figure 3) :

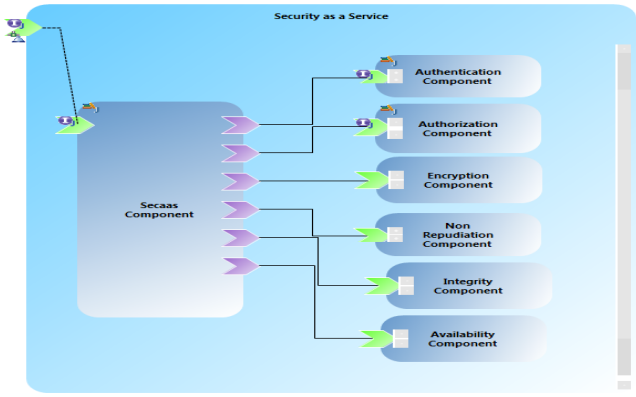

Figure 3: Le composite **Security as a Service**.

- Le composant *SecaaS* est le point d'entrée du composite. Il reçoit du composant *Mediator* les politiques de sécurité à appliquer. Il est chargé d'analyser ces politiques afin d'identifier le type de service de sécurité (authentification, autorisation, etc.) à appeler.
- Le composant *Authentification* est utilisé pour prouver l'identité d'un utilisateur (humain ou autre service). Ce composant reçoit du composant *SecaaS* la règle de politique à appliquer, extrait les informations sur le patron de sécurité et invoque le mécanisme de sécurité à appliquer. Il peut s'agir d'un mécanisme d'authentification faible de type login/password ou d'une authentification forte de type One Time Password (OTP). Ce composant d'authentification comprend des sous composants comme le composant *SSORegistry* (SSO pour Single Sign On) utilisé pour stocker les informations sur les sessions d'authentification et permettre de récupérer les informations des utilisateurs sans avoir besoin de relancer une authentification.
- Le composant *Authorization* permet de gérer les accès à une ressource ou service

et permet d'approuver ou non l'accès de l'utilisateur au service. Tout comme le composant d'authentification, il reçoit la règle de politique de sécurité et invoque la stratégie d'autorisation à appliquer. La stratégie d'authentification pouvant être une autorisation à base de rôle RBAC implémenté selon le protocole XACML ou une autorisation de type Access Control List (ACL).
- Le composant *Encryption* assure le chiffrement et le déchiffrement des données et messages qui lui ont été transmis. Il assure aussi le transport sécurisé en utilisant des protocoles de communication sécurisée (SSL).
- Le composant *Integrity* garantit l'intégrité des données et messages échangés par l'application de fonctions de signature et de hachage de messages.
- Le composant *NonRepudiation* est chargé d'enregistrer les actions des utilisateurs (authentification, accès à un service ou une donnée, modification / destruction d'une donnée, etc.). Ces informations peuvent ensuite être utilisées à des fins d'audit et de surveillance.
- Le composant *Availability* est responsable de la disponibilité des services en fournissant l'accès au service ou à un clone (service redondant) de celui-ci si le service cible initial est indisponible. Ce composant assure également la sauvegarde (backup) de données liées aux services afin de permettre la restauration du système en cas d'incident.

Les composants *Encryption*, *Integrity* et *NonRepudiation* peuvent faire appel à des protocoles de sécurité de type WS-Security tels que XML Encryption et XML Signature afin d'assurer le chiffrement et la signature des messages échangés.

4 Exemple d'application

Pour illustrer notre approche, nous proposons un cas simple d'accès à un service de gestion de dossiers d'étudiants nécessitant une authentification ainsi qu'une autorisation. Pour cela, lors de la mise en œuvre de notre approche MDS à base de patrons, les politiques de sécurité sont générées automatiquement (voir figure 4) puis sont associées au service « GestionEtudiant » (voir figure 6).

```
1.  <policies>
2.      <policy id="1" resource="/GestionEtudiant/ConsulterDossier" type="Authentication">
3.          <policyRule >
4.              <pattern layers="Service" metric="0.25" name="LoginPWD" type="Technique">
5.                 <setting key="userRegistry" value="Registry/Users.xml"/>
6.              </pattern>
7.          </policyRule>
8.      </policy>
9.     <policy id="2" resource="/GestionEtudiant/ConsulterDossier" type="Authorization">
10.         <context type="Temporal" value="true"/>
11.         <policyRule >
12.             <pattern layers="Service" metric="0.25" name="ACL" type="Technique">
13.                 <setting key="accessFile" value="resources/acl/AccessControlList.xml"/>
14.             </pattern>
15.         </policyRule>
16.     </policy>
17. </policies>
```

Figure 4: Les politiques de sécurité associées à la ressource « GestionEtudiant ».

Ces politiques spécifient que ce service possède une opération nommée « ConsulterDossier », qui est considérée comme une ressource (ligne 2 figure 4). Cette ressource nécessite

une authentification (lignes 2 à 8) par login/password (ligne 4) avec la référence au fichier de vérification de l'utilisateur spécifiée à la ligne 5. En plus de l'authentification, un contrôle d'accès (lignes 9 à 16) de type ACL (ligne 12) avec des contraintes horaires (ligne 10) doit être appliqué à cette ressource. La figure 5 décrit le contenu du fichier de vérification des autorisations. Elle montre que les utilisateurs ne peuvent accéder à la ressource qu'à des horaires bien précis (entre 6h et 12h pour l'utilisateur « user1 »).

```
1.  <acl xsi:noNamespaceSchemaLocation="AccessListSchema.xsd">
2.      <resource name="/GestionEtudiant/ConsulterDossier">
3.          <grant user="user1@insa-lyon.fr" beginTime="06:00:00" endTime="12:00:00"></grant>
4.          <grant user="user27@insa-lyon.fr" beginTime="06:00:00" endTime="20:00:00"></grant>
5.      </resource>
6.  </acl>
```

Figure 5: Le fichier des autorisations « AccessControlList.xml ».

Ces politiques de sécurité, qui représentent en fait notre modèle de sécurité à l'exécution, sont associées à la description du service (voir figure 6) en incluant la référence au fichier de politique (ligne 3). Cela permet à l'exécution du service métier d'identifier les politiques de sécurité à appliquer.

```
1.  <wsdl:binding name="GestionEtudiantSoapBinding" type="tns:GestionEtudiantServicePortType">
2.      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
3.        <wsdl:operation name=" ConsulterDossier" mds:policyRef="resources/data/policies.xml"
4.  xmlns:mds="http://mds.org">
5.        <soap:operation soapAction="" style="document"/>
6.        <wsdl:input name="ConsulterDossier">
7.          <soap:body use="literal"/>
8.        </wsdl:input>
9.        <wsdl:output name="ConsulterDossierResponse">
10.         <soap:body use="literal"/>
11.       </wsdl:output>
12.     </wsdl:operation>
13. </wsdl:binding>
```

Figure 6: L'association du fichier de politique à l'opération *ConsulterDossier* du service WSDL *GestionEtudiant*.

```
1.  <composite xmlns="http://www.osoa.org/xmlns/sca/1.0" xmlns:cxf="org/ow2/frascati/intent/cxf"
2.  xmlns:frascati="http://frascati.ow2.org/xmlns/sca/1.1"
3.  xmlns:wsdli="http://www.w3.org/2004/08/wsdl-instance" name="gestionEtudiant" >
4.    <service name="GestionEtudiant" promote="GestionEtudiantComponent/GestionEtudiant">
5.      <interface.java interface="etudiant.api.gestionEtudiantService"/>
6.      <binding.ws requires="MDSatRuntime" uri="/gestionEtudiant-ws-mds"
7.  wsdlElement="http://api.etudiant/#wsdl.port(GestionEtudiantService/GestionEtudiantServicePort)"
8.  wsdli:wsdlLocation="resources/wsdl/gestionEtudiant.wsdl"/>
9.      <frascati:binding.rest requires="MDSatRuntime" uri="/gestionEtudiant-rest-mds"/>
10.   </service> ….
11.  </composite>
```

Figure 7: L'association du composant *GestionEtudiant* avec le composant **MDS@Runtime**.

Afin d'appliquer cette politique de sécurité à l'exécution, le binding Web Service (ligne

6 figure 7) du service « GestionEtudiant » (ligne 4) nécessite d'être associé au composite **MDS@Runtime**[2]. Cette spécification permet d'intercepter les appels à ce service et d'invoquer le composite **MDS@Runtime** avant d'invoquer le service métier lui-même.

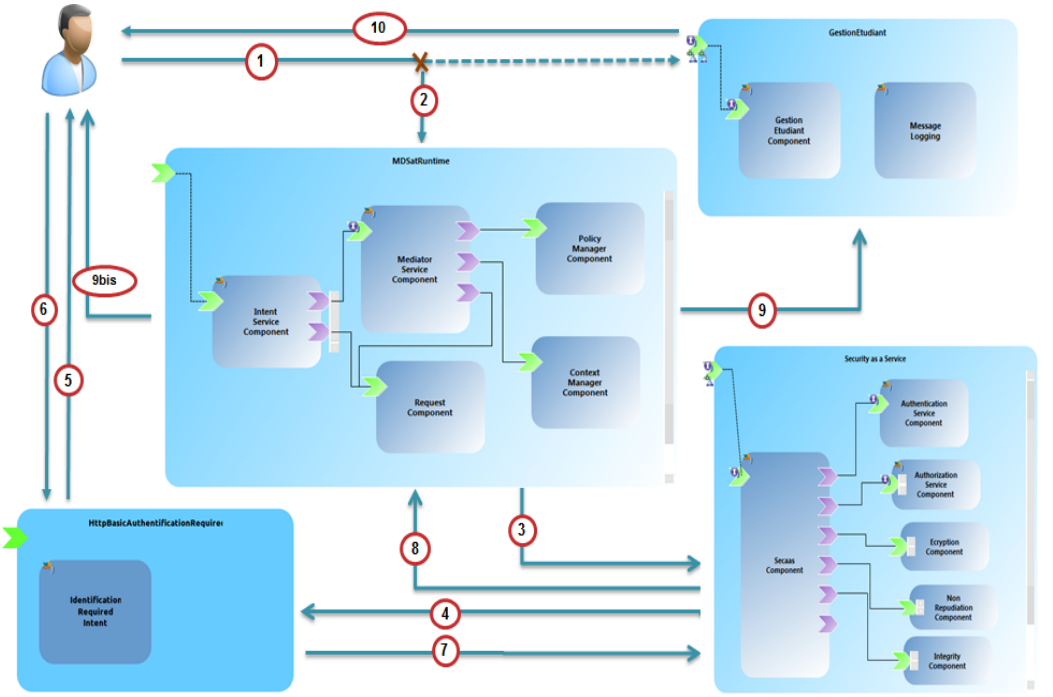

Figure 8: Le processus d'exécution du service « GestionEtudiant » incluant l'exécution de **MDS@Runtime**.

La figure 8 illustre le processus d'exécution complet. L'appel du service « GestionEtudiant » par l'utilisateur (1) est intercepté par le composant *Intent* du composite **MDS@Runtime** (2) qui procède au formatage du message Apache CXF intercepté pour initialiser le composant *Request*. Le composant *Intent* invoque le composant *Mediator* qui récupère auprès du composant *Request* les informations sur le service (nom et opération du service invoqué ainsi que la description WSDL du service). Ensuite, le *Mediator* parse la description du service afin de récupérer la politique de sécurité associée au service. La référence de la politique de sécurité est passée au composant *PolicyManager* qui retourne au *Mediator* la liste des politiques de sécurité à appliquer. Le *Mediator* procède ensuite à l'orchestration des politiques. Il appelle le composant *SecaaS* qui applique les politiques orchestrées (3). Pour cela, le composant *SecaaS* analyse la politique, identifie le service de sécurité correspondant puis l'invoque. Dans notre exemple, c'est le service d'authentification par login/password qui est d'abord invoqué. Si l'utilisateur n'est pas encore connecté, il sera redirigé vers un service de fournisseur d'identité (4) (le service *HttpBasicAuthentification*) afin de recueillir l'identité de l'utilisateur (5) et (6). Cette identité est ensuite

2. L'attribut XML *requires* est le moyen en FraSCAti de tisser un aspect sur une construction SCA.

renvoyée (7) au service d'authentification qui effectue la vérification. Si la vérification est positive, un token d'authentification est alors généré. Ce token contient un identifiant unique qui garantit qu'il y a eu authentification de l'utilisateur. Il permet aussi de retrouver l'identité réelle de l'utilisateur ainsi que de le ré authentifier automatiquement à la manière de SSO lors de l'accès à d'autres services. Le résultat de la vérification est ensuite renvoyé au *Mediator* (8) qui poursuit l'orchestration des politiques tant que celles-ci se terminent par un succès avant d'invoquer le service métier. Dans notre exemple, après l'authentification, c'est le service d'autorisation qui sera invoqué. En cas de réussite (autorisation accordée), l'accès au service métier est accordé et ce service métier est enfin invoqué (9) (10), sinon un message d'erreur est renvoyé à l'utilisateur (9bis).

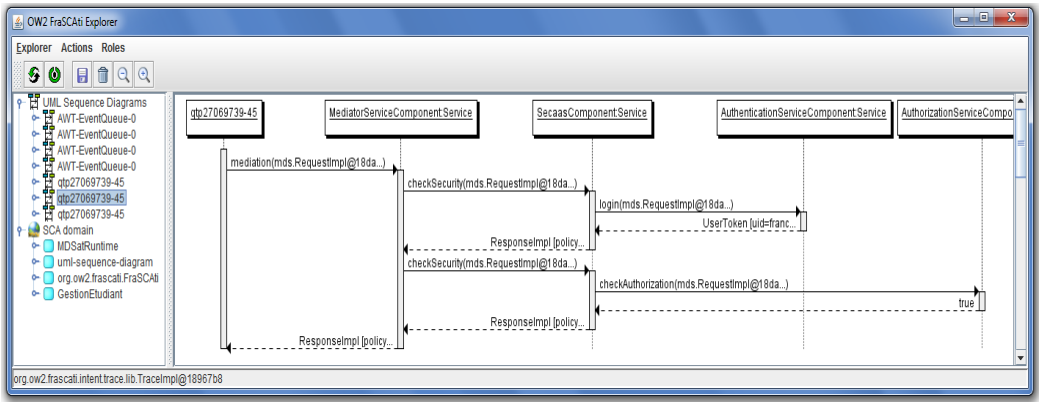

Figure 9: Interactions à l'intérieur du composite **MDS@Runtime**.

Grâce à l'outil FraSCAti Explorer [SMF+09], il est possible de suivre l'enchainement des invocations à l'intérieur du composite **MDS@Runtime** sous la forme d'un diagramme de séquence UML comme illustré en figure 9.

5 Conclusion

La sécurisation de processus métiers déployés dans les nuages passe par la prise en compte du contexte métier du processus mais aussi du contexte de déploiement et d'exécution. Afin d'assurer la protection des services du processus, nous proposons d'étendre l'approche MDS jusqu'à la phase d'exécution afin de sélectionner à l'exécution uniquement les politiques de sécurité qui doivent s'appliquer au contexte. Notre architecture de sécurité, conçue comme un service pluggable sur l'intergiciel FraSCAti, intercepte à la volée les interactions du client avec les services et orchestre les services de sécurité correspondants aux besoins spécifiés dans les politiques de sécurité en fonction du contexte de déploiement. Notre implémentation avec FraSCAti permet d'utiliser notre service de sécurisation pour un déploiement de processus collaboratifs dans un environnement multi-nuages. L'agrégation des politiques de sécurité et l'optimisation des invocations aux services de sécurité (éviter les invocations doublons) constituent les prochaines extensions de notre service de gestion de la sécurité à l'exécution (MDS@Runtime).

Références

[BDL03] D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security for Process Oriented Systems. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT 03)*, pages 100–109. ACM, 2003.

[CSBE08] M. Clavel, V. Silva, C. Braga, and M. Egea. Model-Driven Security in Practice : An Industrial Experience. In *4th European Conference on Model Driven Architecture : Foundations and Applications (CMDA-FA 08)*, pages 326–337, 2008.

[Lou08] A. Louis. *Bus de Service ESB, Nouvelle technologie pour l'intégration.* Livre blanc, Petals Link, 2008.

[LS09] U. Lang and R. Schreiner. Model Driven Security Management : Making Security Management Manageable in Complex Distributed Systems. In *Workshop on Modeling Security (MODSEC08) - International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2009.

[MRS11] Philippe Merle, Romain Rouvoy, and Lionel Seinturier. A Reflective Platform for Highly Adaptive Multi-Cloud Systems. In *International Workshop on Adaptive and Reflective Middleware (ARM'11) - 12th ACM/IFIP/USENIX International Middleware Conference*, pages 14–21. ACM, 2011.

[OBG12] Wendpanga Francis Ouedraogo, Frederique Biennier, and Parissa Ghodous. Adaptive Security Policy Model to Deploy Business Process in Cloud Infrastructure. In *2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*, pages 287–290, 2012.

[PHM+12] Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy, and Lionel Seinturier. A Federated Multi-Cloud PaaS Infrastructure. In *5th International Conference on Cloud Computing (CLOUD'12)*, pages 392–399. IEEE, 2012.

[SHLP05] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The Enterprise Service Bus : Making Service Oriented Architecture Real. *IBM Systems Journal*, 44 :781–797, 2005.

[SMF+09] Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, and Jean-Bernard Stefani. Reconfigurable SCA applications with the FraSCAti Platform. In *IEEE International Conference on Services Computing (SCC'09)*, pages 268–275. IEEE, 2009.

[SMR+12] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Software : Practice and Experience*, 42(5) :559–583, 2012.

[SSL+09] A. R. Souza, B. L. Silva, F. A. Lins, J. C. Damasceno, N. S. Rosa, P. R. Maciel, R. W. Medeiros, B. Stephenson, H. R. Motahari-Nezhad, J. Li, and C. Northfleet. Sec-MoSC Tooling - Incorporating Security Requirements into Service Composition. In *7th International Joint Conference on Service-Oriented Computing (ICSOC-ServiceWave 09)*, pages 649–650, 2009.

[WMS+09] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. Model-driven business process security requirement specification. *Journal of Systems Architecture (JSA)*, pages 211–223, 2009.